US 20050138015A1

(19) United States

(12) Patent Application Publication

Dageville et al.

(10) Pub. No.: US 2005/0138015 A1

(43) Pub. Date: Jun. 23, 2005

(54) HIGH LOAD SQL DRIVEN STATISTICS COLLECTION

(75) Inventors: **Benoit Dageville**, Foster City, CA (US); **Mohamed Ziauddin**, Pleasanton, CA (US); **Mohamed Zait**, San Jose, CA (US); **Dinesh Das**, Redwood City, CA (US)

Correspondence Address:
BINGHAM, MCCUTCHEN LLP
THREE EMBARCADERO CENTER
18 FLOOR
SAN FRANCISCO, CA 94111-4067 (US)

(73) Assignee: **ORACLE INTERNATIONAL CORPORATION**, REDWOOD SHORES, CA

(21) Appl. No.: **10/936,427**

(22) Filed: **Sep. 7, 2004**

Related U.S. Application Data

(60) Provisional application No. 60/500,490, filed on Sep. 6, 2003.

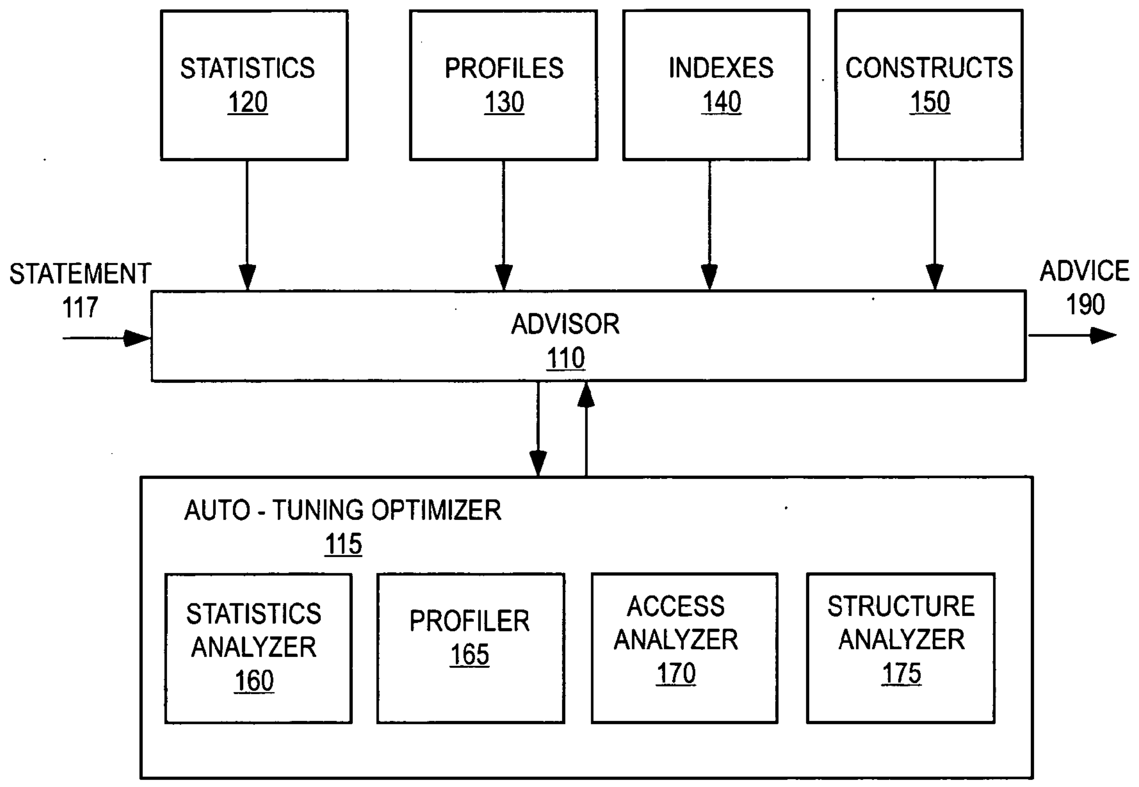
Publication Classification

(51) Int. Cl.^{7} **G06F 17/00**

(52) U.S. Cl. **707/3**

(57) **ABSTRACT**

A method for receiving a database query language statement and statistics information about the statement at an optimizer, and identifying an inaccurate statistic for the statement, is disclosed.



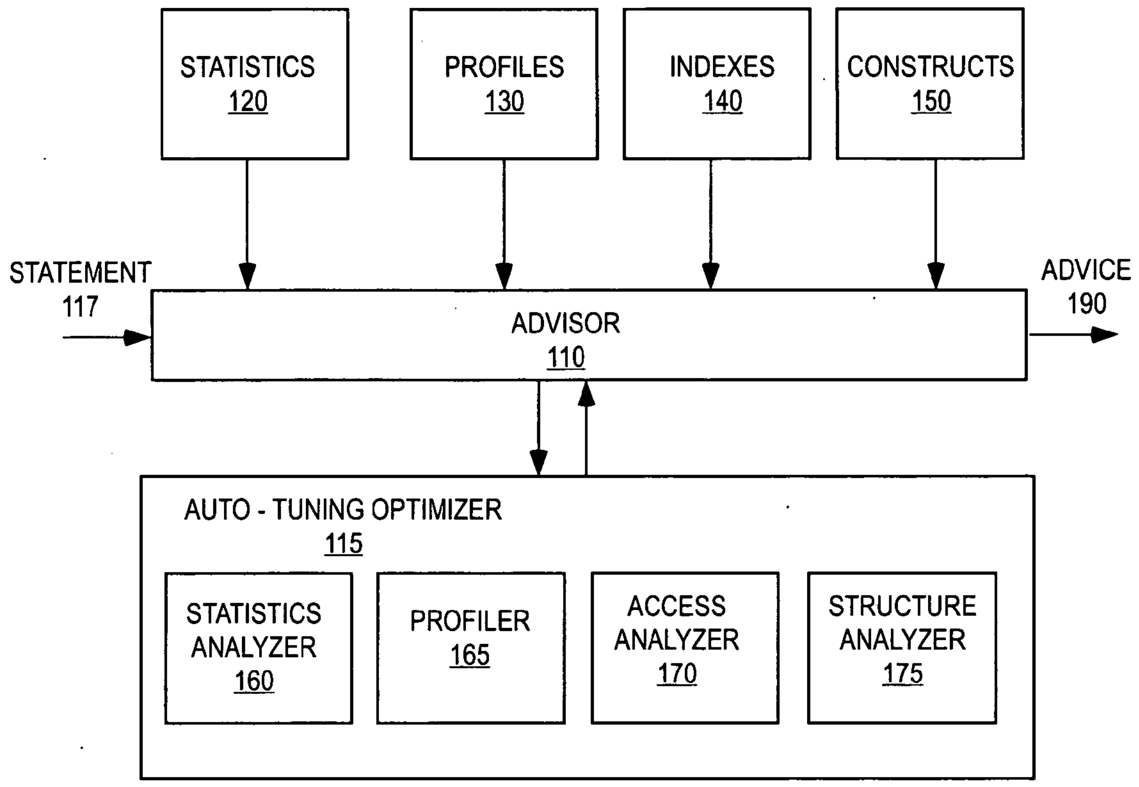

STATISTICS 120
PROFILES 130
INDEXES 140
CONSTRUCTS 150
STATEMENT 117
ADVISOR 110
ADVICE 190
AUTO - TUNING OPTIMIZER 115
STATISTICS ANALYZER 160
PROFILER 165
ACCESS ANALYZER 170
STRUCTURE ANALYZER 175

FIG. 1

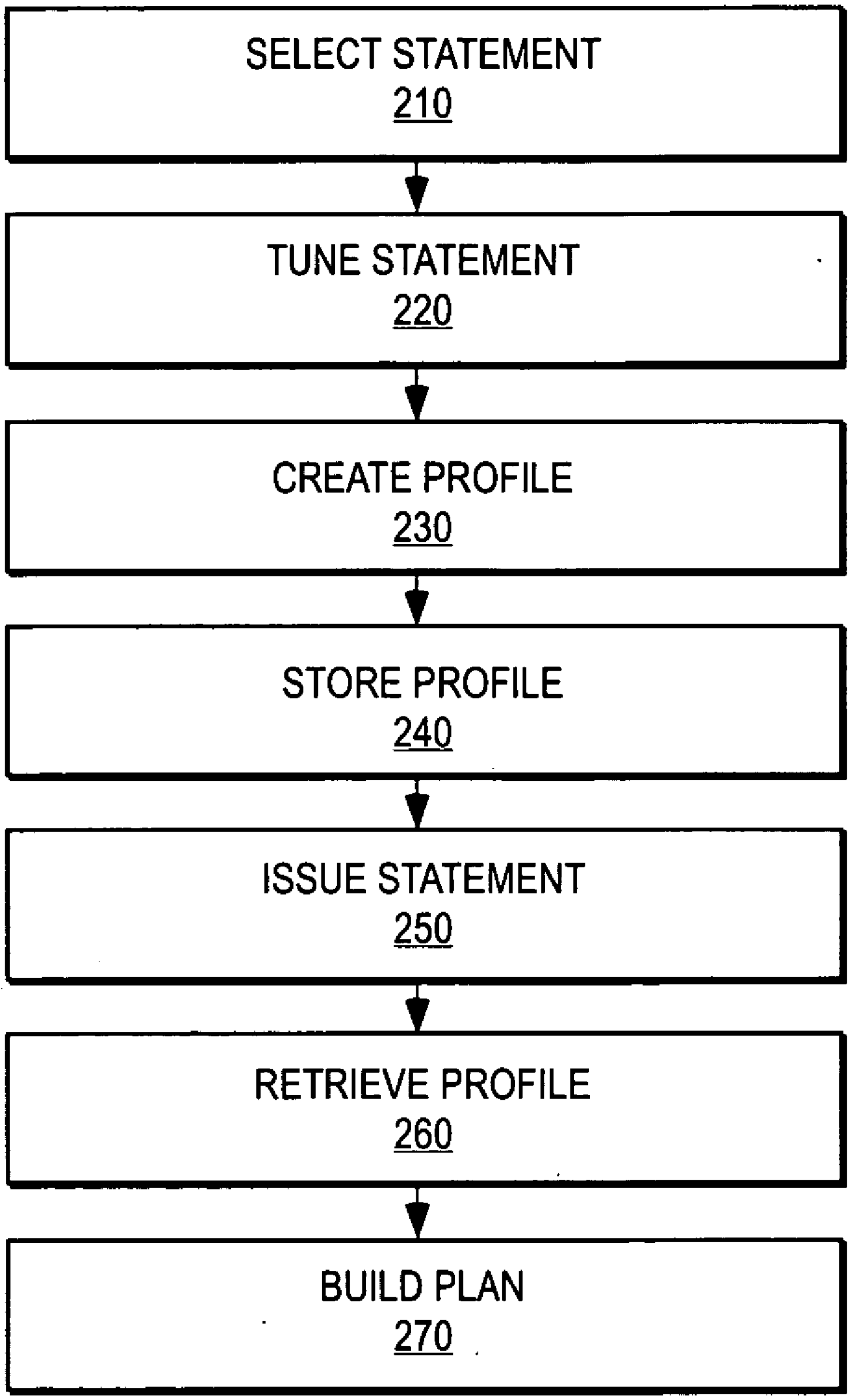

SELECT STATEMENT
210
TUNE STATEMENT
220
CREATE PROFILE
230
STORE PROFILE
240
ISSUE STATEMENT
250
RETRIEVE PROFILE
260
BUILD PLAN
270

FIG. 2

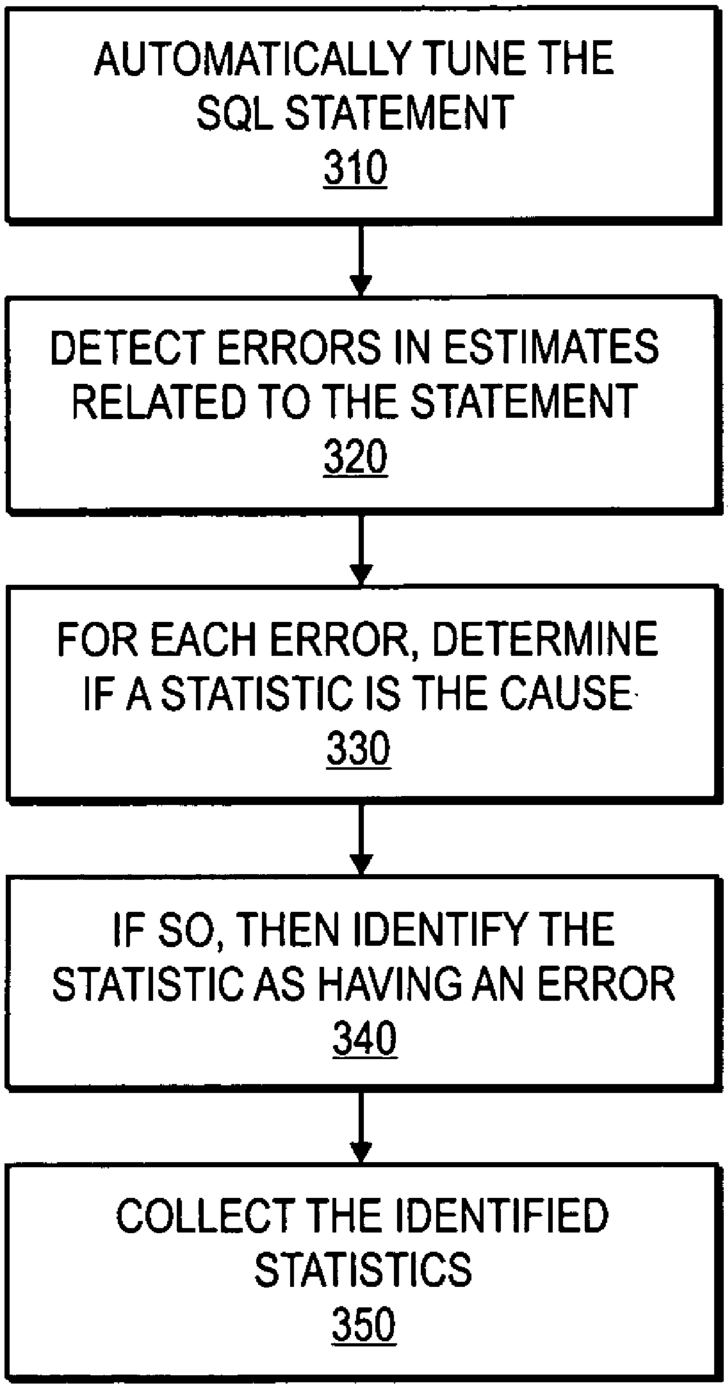

AUTOMATICALLY TUNE THE SQL STATEMENT 310
DETECT ERRORS IN ESTIMATES RELATED TO THE STATEMENT 320
FOR EACH ERROR, DETERMINE IF A STATISTIC IS THE CAUSE 330
IF SO, THEN IDENTIFY THE STATISTIC AS HAVING AN ERROR 340
COLLECT THE IDENTIFIED STATISTICS 350

FIG. 3

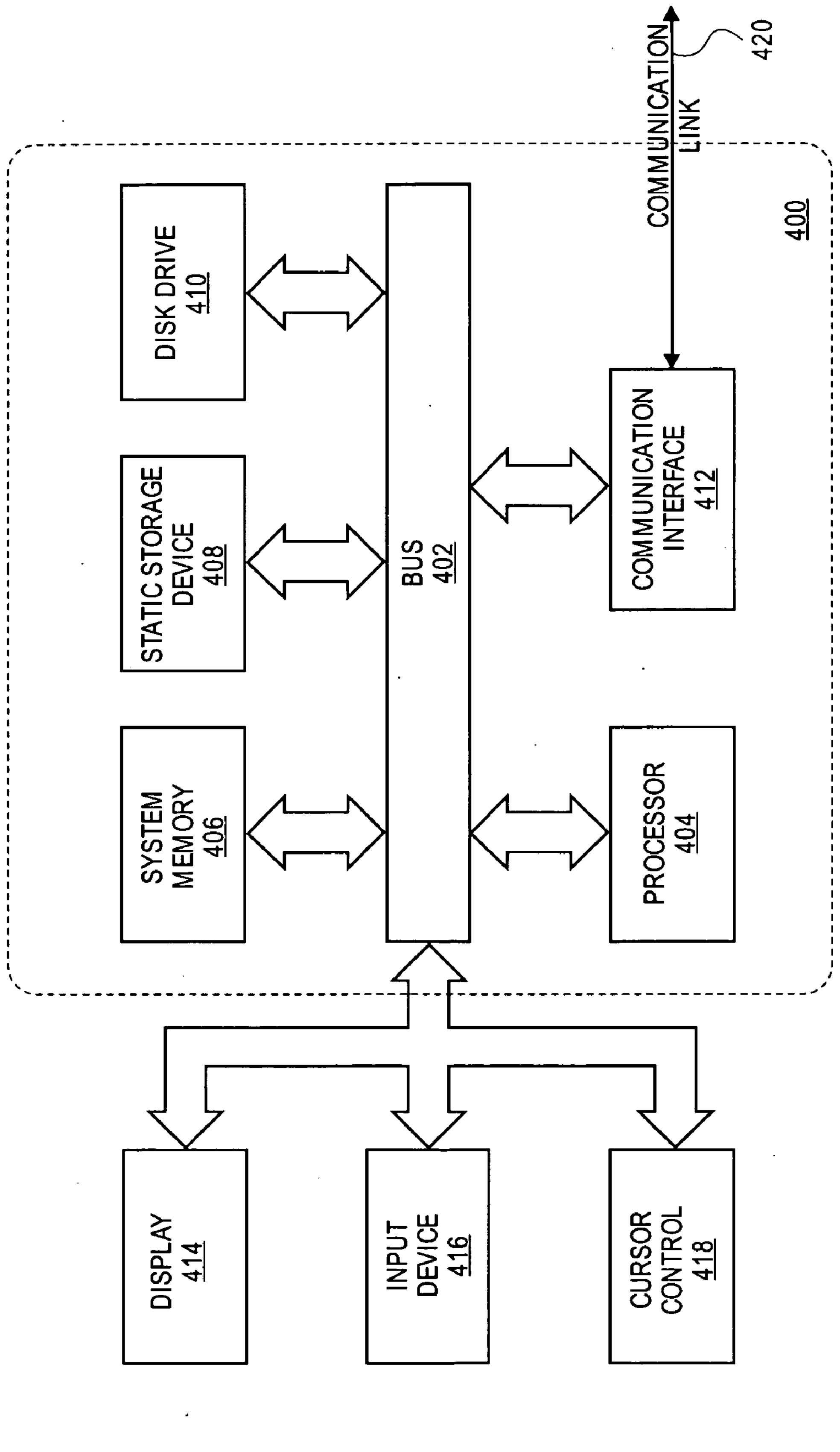

DISK DRIVE 410
STATIC STORAGE DEVICE 408
SYSTEM MEMORY 406
BUS 402
COMMUNICATION LINK
420
COMMUNICATION INTERFACE 412
PROCESSOR 404
400
DISPLAY 414
INPUT DEVICE 416
CURSOR CONTROL 418

FIG. 4

HIGH LOAD SQL DRIVEN STATISTICS COLLECTION

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims the benefit of U.S. Provisional Application No. 60/500,490, filed Sep. 6, 2003, which is incorporated herein by reference in its entirety. This application is related to co-pending applications "SQL TUNING SETS," Attorney Docket No. OI7036272001; "AUTO-TUNING SQL STATEMENTS," Attorney Docket No. OI7037042001; "SQL PROFILE," Attorney Docket No. OI7037052001; "GLOBAL HINTS," Attorney Docket No. OI7037062001; "SQL TUNING BASE," Attorney Docket No. OI7037072001; "AUTOMATIC LEARNING OPTIMIZER," Attorney Docket No. OI7037082001; "AUTOMATIC PREVENTION OF RUN-AWAY QUERY EXECUTION," Attorney Docket No. OI7037092001; "METHOD FOR INDEX TUNING OF A SQL STATEMENT, AND INDEX MERGING FOR A MULTI-STATEMENT SQL WORKLOAD, USING A COST-BASED RELATIONAL QUERY OPTIMIZER," Attorney Docket No. 017037102001; "SQL STRUCTURE ANALYZER," Attorney Docket No. OI7037112001; "AUTOMATIC SQL TUNING ADVISOR," Attorney Docket No. OI7037132001, all of which are filed Sep. 7, 2004 and are incorporated herein by reference in their entirety.

FIELD OF THE INVENTION

[0002] This invention is related to the field of electronic database management systems.

BACKGROUND

[0003] SQL tuning is a very critical aspect of database performance tuning. Typically the database administrator (DBA) or an application developer performs the tuning process. However, it is often a very challenging task. First, it requires a high level of expertise in several complex areas: query optimization, access design, and SQL design. Second, it is a time consuming process because each statement is unique and needs to be tuned individually. Third, it requires an intimate knowledge of the database as well as the application. Finally, the SQL tuning activity is a continuous task because the SQL workload and the database are always changing.

[0004] For example, because the query optimizer relies on statistics of the query objects to function properly, it is important that proper statistics are gathered and kept up to date upon data and schema changes. The problem is that the statistics are potentially expensive to collect. This is especially true if complex data statistics are supported by the database system, like multi-column histograms, either within or across tables. In this context, it is unrealistic to collect all possible data statistics since the number is virtually unlimited. Even without complex statistics, few database systems can afford to collect histogram statistics on every skewed column.

[0005] Hence only a subset of the statistics can generally be collected for a given database system and the problem is to determine what is this appropriate subset. It is defined as the one which gives the best improvement quality of execution plans while, at the same time, being fast enough to collect (generally the DBA can dedicate at most few hours per day or per week to refresh optimizer statistics). It is unrealistic to expect that the DBA or the application developer will be able to determine this appropriate subset since he will have to know which data statistics are really needed by the query optimizer for his database system. This problem is even worse since the appropriate subset might fluctuate overtime as the SQL workload changes.

SUMMARY

[0006] A method for receiving a database query language statement and statistics information about the statement at an optimizer, and identifying an inaccurate statistic for the statement, is disclosed.

BRIEF DESCRIPTION OF DRAWINGS

[0007] **FIG. 1** shows the Automatic SQL Tuning architecture used in the automatic collection process.

[0008] **FIG. 2** shows the process flow of the creation and use of a SQL Profile for the process.

[0009] **FIG. 3** represents an illustration of the automatic statistics collection process.

[0010] **FIG. 4** is a block diagram of a computer system suitable for implementing an nt of the automatic statistics collection process.

DETAILED DESCRIPTION

[0011] Overview

[0012] The embodiments of the invention are described using the term "SQL", however, the invention is not limited to just this exact database query language, and indeed may be used in conjunction with other database query languages and constructs.

[0013] To automatically determine which set of statistics would help the query optimizer improve the execution plan for a SQL statement, each high load query is auto-tuned with an automatic SQL tuning optimizer as shown in **FIG. 1**. Every time an error is made when estimating the cardinality of an intermediate result, the SQL-driven statistics collection component attempts to determine the cause of that mistake, i.e. the process determines if a missing statistic is causing that particular mistake. For example, if the query has a predicate t1.c1.=5 and the optimizer makes a mistake when it computes the selectivity of the predicate, the SQL-driven statistics collection component will determine that a histogram is missing on column t1.c1. A new entry will be created for that histogram. The process will also record the query associated with the statistic, plus compute an estimate of the time to create that histogram.

[0014] This process will be repeated for every query in the SQL tuning set. At the end, the maximum set of statistic objects to create is analyzed. If the sum of the estimated creation time for these objects is less than the time constraint, all these statistics can be collected. Otherwise only a subset is collected. The subset can be determined using a knapsack based approach. The weight of each object is the time to create a statistic object while the benefit of creating the statistic object is computed based on the overall improvement of the SQL workload.

[0015] Automatic SQL Tune Advisor Architecture

[0016] The Automatic SQL tuning process is implemented by the Automatic Tuning Optimizer, which performs several tuning analyses during the process. The output of a tuning analysis is a set of tuning recommendations, which may be presented to the user. **FIG. 1** shows the Automatic SQL Tuning architecture and the functional relationship between its two sub-components. The SQL tune advisor **110** receives a SQL statement **117** and information to tune the statement. The information can be related to missing or stale statistics, **120**. The information may be a set of tuning hints stored in SQL profile **130**. The information may be related to missing indexes **140**. Also, the information may be about the construct of the SQL statement **150**. The tune advisor **110** provides the statement and its related information to auto-tuning optimizer **115**. The optimizer automatically tunes the statement with the information by performing auto-tuning processes. The statistics analyzer **160** generates adjustment factors to correct missing or stale statistics. The SQL profiler **165** generates tuning hints for the statement in the form of a profile. The access path analyzer **170** generates indexes. The SQL structure analyzer **175** generates recommendations for rewriting the statement.

[0017] SQL Profiling

[0018] A profiling process is performed by the optimizer during the tuning process to adjust statistics that are used in generating an execution plan for a SQL statement. The profiling process verifies that statistics are not missing or stale, validates the estimates made by the query optimizer for intermediate results, and determines the correct optimizer settings. The Automatic Tuning Optimizer builds a SQL Profile from the tuning information it generates during the statistics analysis (e.g., provides missing statistics for an object), validation of intermediate results estimate, and detection of the best setting for optimizer parameters. When a SQL Profile is built, the Automatic Tuning Optimizer generates a user recommendation to accept a SQL profile.

[0019] Statistics Analysis

[0020] The goal of statistics analysis is to verify that statistics are not missing or stale. The query optimizer logs the types of statistics that are actually used during the plan generation process, in preparation for the verification process. For example, when a SQL statement contains an equality predicate, it logs the column number of distinct values, whereas for a range predicate it logs the minimum and maximum column values information.

[0021] Once the logging of used statistics is complete, the query optimizer checks if each of these statistics is available on the associated query object (i.e. table, index or materialized view). If the statistic is available then it verifies whether the statistic is up-to-date. To verify the accuracy of a statistic, it samples data from the corresponding query object and compares it to the statistic.

[0022] If a statistic is found to be missing, the query optimizer will generate auxiliary information to supply the missing statistic. If a statistic is available but stale, it will generate auxiliary information to compensate for staleness.

[0023] Estimates Analysis

[0024] One feature of a cost-based query optimizer is its ability to derive the size of intermediate results. For example, the optimizer estimates the number of rows from applying table filters when deciding which join algorithm to pick.

[0025] One factor that causes the optimizer to generate a sub-optimal plan is wrong estimate of intermediate result sizes. Wrong estimates can be caused by a combination of the following factors: The predicate (filter or join) is too complex to use standard statistical methods to derive the number of rows (e.g., the columns are compared thru a complex expression like (a*b)/c=10), The data distribution of the column used in the predicate is skewed, and there is no histogram, leading the optimizer to assume a uniform data distribution, or The data in column values is correlated but the optimizer is not aware of it, causing the optimizer to assume data independence.

[0026] During SQL Profiling, the Automatic Tuning Optimizer validates the estimates made by the query optimizer, and compensates for missing information or wrong estimates. The validation process may involve running part of the query on a sample of the input data.

[0027] Parameter Settings Analysis

[0028] The Automatic Tuning Optimizer uses the past execution history of a SQL statement to determine the correct optimizer settings. For example, if the execution history shows that a SQL statement is only partially executed in the majority of times then the appropriate setting will be to optimize it for first n rows, where n is derived from the execution history. This constitutes a customized parameter setting for the SQL statement. (Note that past execution statistics are available in the Automatic Workload Repository (AWR) presented later).

[0029] SQL Profile

[0030] The tuning information produced from the above three analyses is stored in a SQL Profile. Once a SQL Profile is created, it is used in conjunction with the existing statistics by the compiler to produce a well-tuned plan for the corresponding SQL statement. **FIG. 2** shows the process flow of the creation and use of a SQL Profile. The process can have two separate phases: an Automatic SQL Tuning phase, and a regular optimization phase. During the Automatic SQL Tuning phase, a DBA selects a SQL statement **210** and runs the SQL Tuning Advisor. The SQL Tuning Advisor invokes the Automatic Tuning Optimizer to generate tuning recommendations, **220**. The Automatic Tuning Optimizer generates a SQL Profile along with other recommendations, **230**. After a SQL Profile is built, it is stored in the data dictionary, once it is accepted by the user, **240**.

[0031] Later, during the regular optimization phase, a user issues the same SQL statement, **250**. The query optimizer finds the matching SQL profiles from the data dictionary, **260**, and uses the SQL profile information to build a well-tuned execution plan, **270**. The use of SQL Profiles is completely transparent to the user.

[0032] The creation and use of a SQL Profile doesn't require changes to the application source code. Therefore, SQL profiling provides a way to tune SQL statements issued from packaged applications where the users have no access to or control over the application source code.

[0033] Automatic Statistic Collection Process

[0034] The automatic statistics collection process tunes high load SQL statements in a SQL tuning set using the automatic SQL Tuning optimizer. The optimizer automatically tunes each SQL statement by profiling it and by recommending other tuning actions to the end user. **FIG. 3** represents an illustration of the SQL tuning process. To determine which set of statistics would help the query optimizer, each query in the SQL tuning set is first auto-tuned with an automatic SQL tuning optimizer, **310**. Every time an error is made when estimating the cardinality of an intermediate result, **320**, the SQL-driven statistics collection component attempts to determine the cause of that mistake, i.e. the process determines if a missing statistic is causing that particular mistake, **330**. For example, if the query has a predicate t1.c1.=5 and the optimizer makes a mistake when it computes the selectivity of the predicate, the SQL-driven statistics collection component will determine that a histogram is missing on column t1.c1. A new entry in a list of missing or erroneous statistics will be created for that histogram, **340**. The process will also record the query associated with the statistic, plus compute an estimate of the time to create that histogram, **350**.

[0035] This process will be repeated for every query in the SQL tuning set. At the end, the maximum set of statistic objects to create is analyzed. If the sum of the estimated creation time for these objects is less than the time constraint, all these statistics can be collected, **350**. Otherwise only a subset is collected. The subset can be determined using a knapsack based approach. The weight of each object is the time to create a statistic object while the benefit of creating the statistic object is computed based on the overall improvement of the SQL workload.

[0036] Accepting SQL profile recommendations closes an iteration of the SQL tuning loop; SQL profiling will most likely improve the execution plan of the targeted set of SQL statements, hence reducing their overall performance impact on the system. This will be reflected in the performance measurements being collected. The next tuning cycle can then begin with a different set of high-load SQL statements. The process can be repeated several times until the desired performance level is achieved.

[0037] The Automatic SQL Tuning process, which is integrated with the query optimizer, provides a manageability framework for a self-managing database to automatically collect statistics for high-load SQL statements. The Automatic SQL Tuning process tunes SQL statements and produces a set of comprehensive tuning recommendations. In addition to recommendations, it may also build a SQL Profile to store tuning hints for the statement. The user may decide whether to accept the recommendations. Once a SQL Profile is created, the query optimizer will use it to generate a well-tuned plan for the corresponding SQL statement. A tuning object called the SQL Tuning Set provides a store for a SQL workload to be automatically tuned. With the automatic tuning process, automatic tuning results can scale over a large number of queries and can evolve over time with changes in the application workload and the underlying data. Automatic SQL tuning is also far cheaper than manual tuning. Together, these reasons position automatic SQL tuning as an effective and economical alternative to manual tuning.

[0038] **FIG. 4** is a block diagram of a computer system **400** suitable for implementing an embodiment of automatically collecting statistics to improve the execution plan of a high load SQL statement. Computer system **400** includes a bus **402** or other communication mechanism for communicating information, which interconnects subsystems and devices, such as processor **404**, system memory **406** (e.g., RAM), static storage device **408** (e.g., ROM), disk drive **410** (e.g., magnetic or optical), communication interface **412** (e.g., modem or ethernet card), display **414** (e.g., CRT or LCD), input device **416** (e.g., keyboard), and cursor control **418** (e.g., mouse or trackball).

[0039] According to one embodiment of the invention, computer system **400** performs specific operations by processor **404** executing one or more sequences of one or more instructions contained in system memory **406**. Such instructions may be read into system memory **406** from another computer readable medium, such as static storage device **408** or disk drive **410**. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions to implement the invention.

[0040] The term "computer readable medium" as used herein refers to any medium that participates in providing instructions to processor **404** for execution. Such a medium may take many forms, including but not limited to, non-volatile media, volatile media, and transmission media. Non-volatile media includes, for example, optical or magnetic disks, such as disk drive **410**. Volatile media includes dynamic memory, such as system memory **406**. Transmission media includes coaxial cables, copper wire, and fiber optics, including wires that comprise bus **402**. Transmission media can also take the form of acoustic or light waves, such as those generated during radio wave and infrared data communications.

[0041] Common forms of computer readable media includes, for example, floppy disk, flexible disk, hard disk, magnetic tape, any other magnetic medium, CD-ROM, any other optical medium, punch cards, paper tape, any other physical medium with patterns of holes, RAM, PROM, EPROM, FLASH-EPROM, any other memory chip or cartridge, carrier wave, or any other medium from which a computer can read.

[0042] In an embodiment of the invention, execution of the sequences of instructions to practice the invention is performed by a single computer system **400**. According to other embodiments of the invention, two or more computer systems **400** coupled by communication link **420** (e.g., LAN, PTSN, or wireless network) may perform the sequence of instructions to practice the invention in coordination with one another. Computer system **400** may transmit and receive messages, data, and instructions, including program, i.e., application code, through communication link **420** and communication interface **412**. Received program code may be executed by processor **404** as it is received, and/or stored in disk drive **410**, or other non-volatile storage for later execution.

[0043] In the foregoing specification, the invention has been described with reference to specific embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention. The specification and drawings are, accordingly, to be regarded in an illustrative rather than restrictive sense.

We claim:

1. A method comprising:

receiving a database query language statement and execution statistics information about the statement at an optimizer; and

identifying one or more inaccurate or missing statistics for the statement.

2. The method of claim 1, wherein identifying further comprises:

identifying a problem with an execution plan of each statement; and

determining that one or more inaccurate or missing statistics causes the problem of the execution plan.

3. The method of claim 2, further comprising:

creating an entry for the inaccurate or missing statistic in a collection list.

4. The method of claim 3, further comprising:

automatically collecting statistics for each entry in the collection list.

5. The method of claim 1, wherein receiving the statement further comprises:

identifying high load statements;

storing the high load statements in a tuning set; and

sending each high load statement in the tuning set to the optimizer.

6. The method of claim 1, wherein the database query language statement is a SQL statement.

7. An apparatus comprising:

means for receiving a database query language statement and execution statistics information about the statement at an optimizer; and

means for identifying one or more inaccurate or missing statistics for the statement.

8. The apparatus of claim 7, wherein said means for identifying further comprises:

means for identifying a problem with an execution plan of each statement; and

means for determining that one or more inaccurate or missing statistics causes the problem of the execution plan.

9. The apparatus of claim 8, further comprising:

means for creating an entry for the inaccurate statistic in a collection list.

10. The apparatus of claim 9, further comprising:

means for automatically collecting statistics for each entry in the collection list.

11. The apparatus of claim 7, wherein said means for receiving the statement further comprises:

means for identifying high load statements;

means for storing the high load statements in a tuning set; and

means for sending each high load statement in the tuning set to the optimizer.

12. The apparatus of claim 7, wherein the database query language statement is a SQL statement.

13. A computer readable medium storing a computer program of instructions which, when executed by a processing system, cause the system to perform a method comprising:

receiving a database query language statement and execution statistics information about the statement at an optimizer; and

identifying one or more missing or inaccurate statistics for the statement.

14. The medium of claim 13, wherein identifying further comprises:

identifying a problem with an execution plan of each statement; and

determining that one or more inaccurate or missing statistics causes the problem of the execution plan.

15. The medium of claim 14, wherein the instructions, when executed, further perform the method comprising:

creating an entry for the inaccurate statistic in a collection list.

16. The medium of claim 15, wherein the instructions, when executed, further perform the method comprising:

automatically collecting statistics for each entry in the collection list.

17. The medium of claim 13, wherein receiving the statement further comprises:

identifying high load statements;

storing the high load statements in a tuning set; and

sending each high load statement in the tuning set to the optimizer.

18. The medium of claim 13, wherein the database query language statement is a SQL statement.

* * * * *